N88-17240

CONFIG: QUALITATIVE SIMULATION TOOL FOR ANALYZING BEHAVIOR OF ENGINEERED DEVICES

Jane T. Malin and Bryan D. Basham
Systems Development and Simulation Div.
Engineering Directorate
NASA - Lyndon B. Johnson Space Center
Houston, TX 77058

Richard A. Harris
MITRE Corporation
1120 NASA Rd. 1
Houston, TX 77058

ABSTRACT

To design failure management expert systems, engineers mentally analyze the effects of failures and procedures as they propagate through device configurations. CONFIG is a generic device modelling tool for use in discrete event simulation, to support such analyses. CONFIG permits graphical modeling of device configurations and qualitative specification of local operating modes of device components. Computation requirements are reduced by focussing the level of component description on operating modes and failure modes, and specifying qualitative ranges of variables relative to mode transition boundaries. A time-step approach is avoided, and simulation processing occurs only when modes change or variables cross qualitative boundaries. Device models are built graphically, using components from libraries. Components are connected at ports by graphical relations that define data flow. The core of a component model is its state-transition diagram, which specifies modes of operation and transitions among them. Process statements describe state transitions and within-state processing, and have three parts: invocations (preconditions for effects execution), effects, and delays for each effect. A process language supports writing statements with several qualitative and quantitative syntaxes, including table lookups. CONFIG has been used to build device models in two domains, digital circuits and thermal systems.

INTRODUCTION

Designing, testing, and operating engineered devices requires analysis of the effects of failures and procedures as they propagate through device configurations. Such analysis is required in development of failure management expert systems, in failure modes and effects analysis (FMEA), and in procedures development. Information about device configuration and operating modes is used to predict effects of local changes in components on the device as a whole, and to plan how to diagnose failures and recover from them [1]. Early in design and testing, many of these analyses are performed mentally by engineers on the basis of qualitative information.

The purpose of the CONFIG project is to develop a generic device modeling tool to support commonsense analysis of system behavior by designers and operators. The tool should permit engineers to graphically model device configuration (components and connections) and qualitatively specify local operating modes of components. The tool should provide for integration of models from multiple domains, e.g., electrochemical/thermal processing and digital circuits.

The CONFIG project approach has been to develop both a CONFIG prototype and a prototyping environment. The approach to the CONFIG tool has been developed to support rapid informal analysis early in system design, as well as later precise analysis. The tool permits modeling of component modes and processes both qualitatively and quantitatively. The tool permits graphical modeling of component configurations. The tool is built on a discrete event simulator, and propagates discrete change events through device configurations. The CONFIG project prototyping environment provides flexible object-oriented modeling capabilities and a language constructor that supports experimenting with various qualitative and quantitative representations of device information.

COMPONENT LIBRARIES AND GRAPHICAL MODELING

The CONFIG tool should be designed for ease of use by engineers and operators. As in the work of Towne et al [2], the tool supports the development of graphical libraries of component models, and permits an author to build device models graphically. A model is built by using component objects from a library, and connecting them at ports with graphical relations that define data flow between components. Libraries also define classes of processes and the process language for the author. This aspect of the CONFIG tool uses the Simkit discrete event simulation tool. An example of a thermal model and some component models and relations from its associated library are shown in Figure 1.

COMPONENT DEFINITION

Computation and specification requirements are reduced by focussing the level of component description on operating modes and failure modes, and specifying qualitative ranges of component variables relative to mode transition boundaries.

A component model can be viewed both as a composite of modes and as a composite of ports. The core of a component model is its state-transition diagram, which graphically specifies modes of operation (both normal and failed) and the transitions among them. State transitions and within-state variable transformations are specified as processes. The other decomposition of a component model is into its ports. Ports designate component inputs and outputs. Relations connect ports for data propagation between components. Examples of both types of decomposition are shown in Figure 2. The boxed area on the right of the figure is a decomposition of the boxed area in Figure 1.

DISCRETE EVENT SIMULATION APPROACH

Using an approach similar to Pan [3], discrete events are defined at the level of changes in operating modes. A time-step approach is avoided, and simulation processing need occur only when modes change or variables cross qualitative boundaries.

Rather than constraint propagation, discrete event processes determine the consequences of component changes. The event structure controls the propagation of behavior changes among components. Scheduled events pass data between ports, change variable values, and make state transitions. The primary event is the update of a component, which is triggered by a change in an input variable, local variable, or component state. In such an event, appropriate processes are inspected, and the effects of invoked processes are scheduled with corresponding delays. Updates originating from many components can be scheduled at the same time on the discrete event clock.

PROCESSES AND PROCESS LANGUAGE CONSTRUCTOR

There are three kinds of processes, mode independent processes, mode dependent processes, and mode transition processes (which are actually also mode dependent processes). Processes consist of three parts: invocations (preconditions for effects execution), effects (executed if all invocations are satisfied), and delays corresponding to each effect (effect completions scheduled at increments to the current time). Invocations and effects are defined in terms of variables, modes, and processes.

A key capability is a process language constructor and interpreter that permits process statements to be written with an array of qualitative and quantitative syntaxes, including table lookups. A process language interprets statements that define invocations and effects of processes. The language constructor supports experimenting with representations, by permitting the definition of data-structure types and operators.

Component variables can be specified quantitatively or qualitatively, but a small number of qualitative ranges is desirable (e.g., abnormal-low, low, medium, high, abnormal-high). Continuous behavior is partitioned into trends and breakpoints. The durations of trends can be represented "qualitatively" as an approximate order of magnitude, which is translated into an interval on the discrete event clock. Trend effects are represented by scheduling a process for the end of a trend, at which time the duration of the trend is confirmed before executing the effect.

Examples of processes, data structures and operators from the thermal library are shown in Figures 3 and 4.

CONCLUSION

CONFIG has been used to build device models in two domains, digital circuits and thermal systems. The CONFIG project is ongoing, with plans for additional capabilities beyond the current modeling capabilities. These additional capabilities will include support for analysis of failure effects, and support for development of differential diagnostic measures and tests. An additional goal is to support incremental development of quantitative process simulation algorithms.

ACKNOWLEDGEMENT

The authors wish to thank Jodi Seaborn for developing the thermal library and thermal bus model using CONFIG.

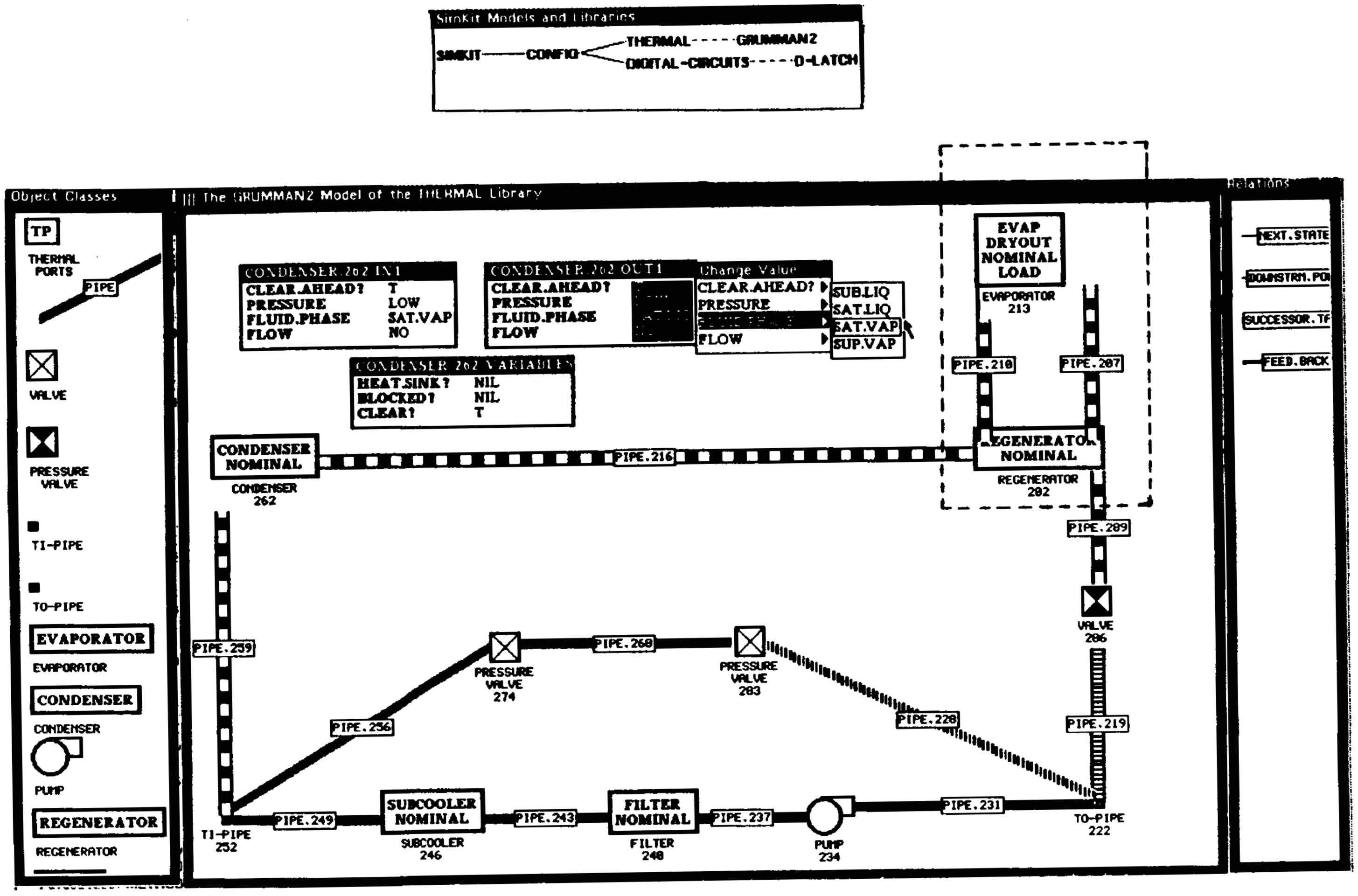
REFERENCES

1. Malin, J. T., and Lance, N., "Processes in Construction of Failure Management Expert Systems from Device Design Information," IEEE TRANS. SYSTEMS, MAN, AND CYBERNETICS, in press.

2. Towne, D. M., Munro, A., Pizzini, Q. A., and Surmon, D. S., "Representing System Behaviors and Expert Behaviors for Intelligent Tutoring," TECH. REPORT NO. 108, Univ. So. Calif. Behavioral Technology Laboratories, Redondo Beach, CA, February, 1987.

3. Pan, J. Y., "Qualitative Reasoning With Deep-level Mechanism Models for Diagnoses of Mechanism Failures," PROC. FIRST CONFERENCE ART. INT. APPLICATIONS, Denver, CO, December, 1984, pp. 295-301.

ORIGINAL PAGE IS
OF POOR QUALITY



CONDENSER.262 IN1

CLEAR.AHEAD?	T
PRESSURE	LOW
FLUID.PHASE	SAT.VAP
FLOW	NO

CONDENSER.262 OUT1

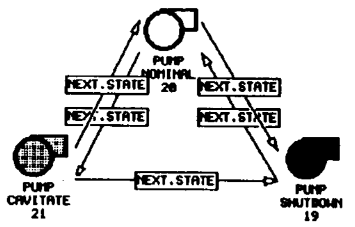
CLEAR.AHEAD?	
PRESSURE	
FLUID.PHASE	
FLOW	

CONDENSER.262 VARIABLES

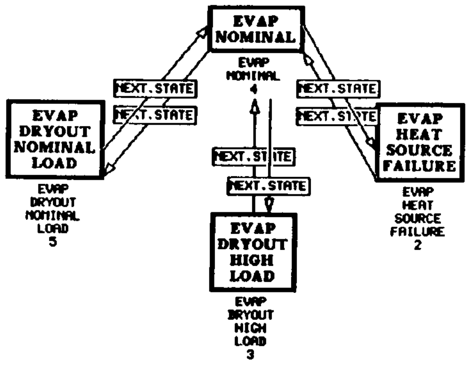
HEAT.SINK?	NIL
BLOCKED?	NIL
CLEAR?	T

Figure 1 - Thermal Library and Grumman Bus Model

State Diagram for Pumps



State Diagram for Evaporators



Decomposition of Evaporator.213, Pipe.207 Regenerator.202, and Pipe.209

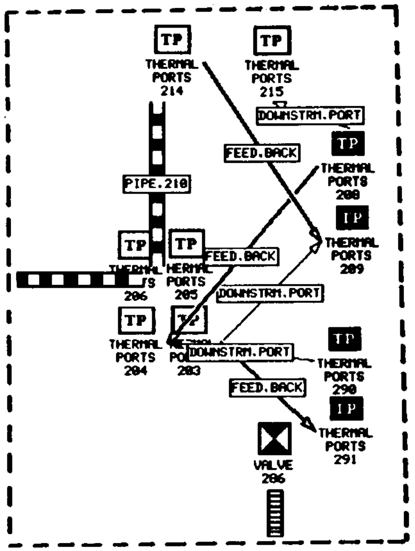
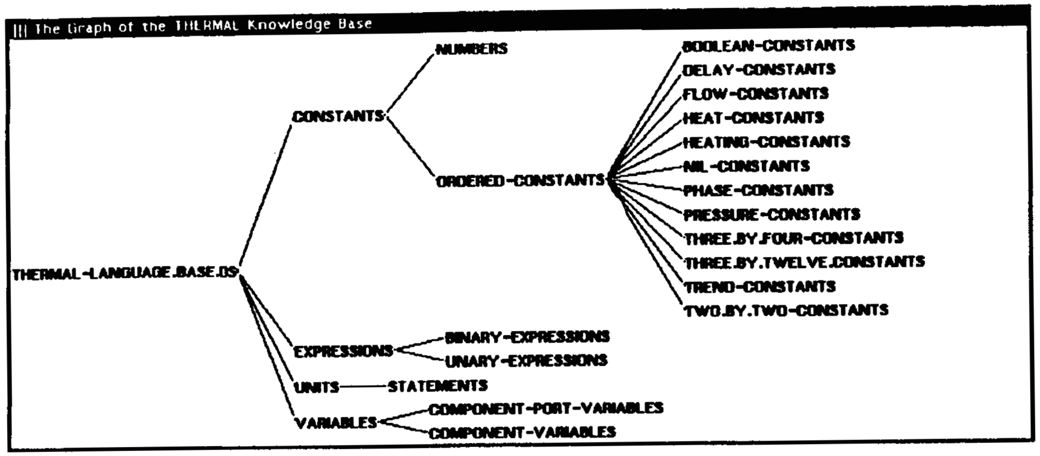
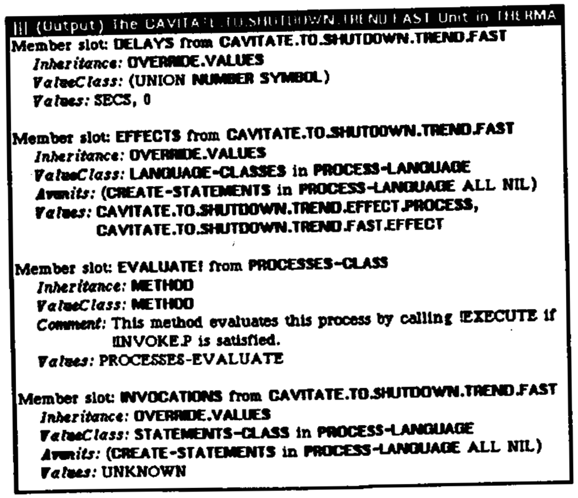


Figure 2 - Thermal Component Decompositions

ORIGINAL PAGE IS OF POOR QUALITY



Process with Delays



(Output) The CAVITATE.TO.SHUTDOWN.TREND.FAST Unit in THERMA

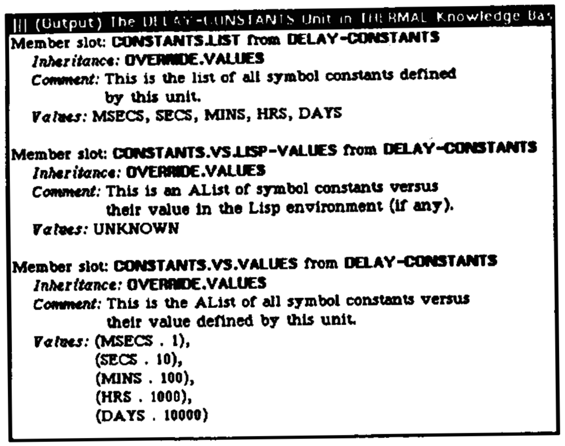
```
Member slot: DELAYS from CAVITATE.TO.SHUTDOWN.TREND.FAST
  Inheritance: OVERRIDE.VALUES
  ValueClass: (UNION NUMBER SYMBOL)
  Values: SECS, 0

Member slot: EFFECTS from CAVITATE.TO.SHUTDOWN.TREND.FAST
  Inheritance: OVERRIDE.VALUES
  ValueClass: LANGUAGE-CLASSES in PROCESS-LANGUAGE
  Avmits: (CREATE-STATEMENTS in PROCESS-LANGUAGE ALL NIL)
  Values: CAVITATE.TO.SHUTDOWN.TREND.EFFECT.PROCESS,
          CAVITATE.TO.SHUTDOWN.TREND.FAST.EFFECT

Member slot: EVALUATE! from PROCESSES-CLASS
  Inheritance: METHOD
  ValueClass: METHOD
  Comment: This method evaluates this process by calling !EXECUTE if
           !INVOKE.P is satisfied.
  Values: PROCESSES-EVALUATE

Member slot: INVOCATIONS from CAVITATE.TO.SHUTDOWN.TREND.FAST
  Inheritance: OVERRIDE.VALUES
  ValueClass: STATEMENTS-CLASS in PROCESS-LANGUAGE
  Avmits: (CREATE-STATEMENTS in PROCESS-LANGUAGE ALL NIL)
  Values: UNKNOWN
```

Delay Constants



(Output) The DELAY-CONSTANTS Unit in THERMAL Knowledge Bas

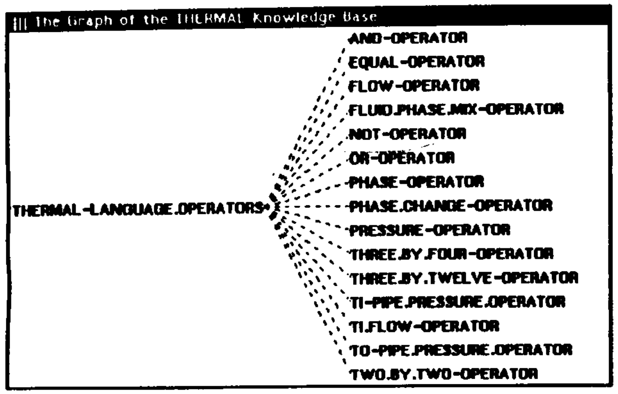
```
Member slot: CONSTANTS.LIST from DELAY-CONSTANTS
  Inheritance: OVERRIDE.VALUES
  Comment: This is the list of all symbol constants defined
           by this unit.
  Values: MSECS, SECS, MINS, HRS, DAYS

Member slot: CONSTANTS.VS.LISP-VALUES from DELAY-CONSTANTS
  Inheritance: OVERRIDE.VALUES
  Comment: This is an AList of symbol constants versus
           their value in the Lisp environment (if any).
  Values: UNKNOWN

Member slot: CONSTANTS.VS.VALUES from DELAY-CONSTANTS
  Inheritance: OVERRIDE.VALUES
  Comment: This is the AList of all symbol constants versus
           their value defined by this unit.
  Values: (MSECS . 1),
          (SECS . 10),
          (MINS . 100),
          (HRS . 1000),
          (DAYS . 10000)
```

Figure 3 - Thermal Language Data Structures

ORIGINAL PAGE IS
OF POOR QUALITY



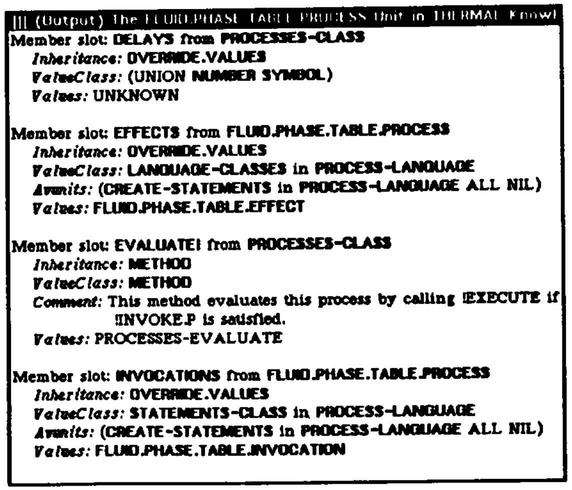
Phase Operator Table

Heat.Grad (component variable)

IN Fluid Phase (port variable)

	neg.high	neg.med	no.change	pos.med	pos.high
sub.liq	sub.liq	sub.liq	sub.liq	sat.liq	sat.vap
sat.liq	sub.liq	sub.liq	sat.liq	sat.vap	sup.vap
sat.vap	sub.liq	sat.liq	sat.vap	sup.vap	sup.vap
sup.vap	sat.liq	sat.vap	sup.vap	sup.vap	sup.vap

Process Using Table


||| (Output) The FLUID.PHASE.TABLE.PROCESS Unit in THERMAL Knowl

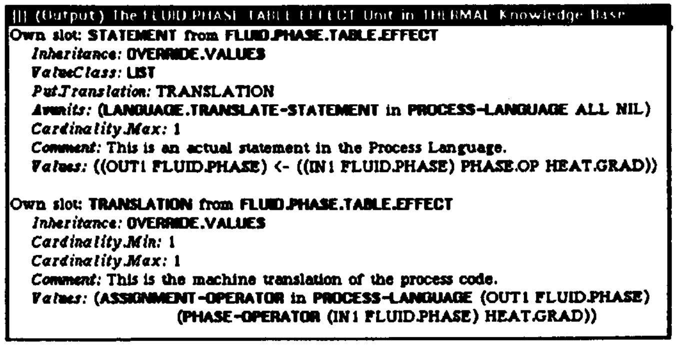
Member slot: DELAYS from PROCESSES-CLASS
  Inheritance: OVERRIDE.VALUES
  ValueClass: (UNION NUMBER SYMBOL)
  Values: UNKNOWN

Member slot: EFFECTS from FLUID.PHASE.TABLE.PROCESS
  Inheritance: OVERRIDE.VALUES
  ValueClass: LANGUAGE-CLASSES in PROCESS-LANGUAGE
  Avenits: (CREATE-STATEMENTS in PROCESS-LANGUAGE ALL NIL)
  Values: FLUID.PHASE.TABLE.EFFECT

Member slot: EVALUATE! from PROCESSES-CLASS
  Inheritance: METHOD
  ValueClass: METHOD
  Comment: This method evaluates this process by calling !EXECUTE if !INVOKE.P is satisfied.
  Values: PROCESSES-EVALUATE

Member slot: INVOCATIONS from FLUID.PHASE.TABLE.PROCESS
  Inheritance: OVERRIDE.VALUES
  ValueClass: STATEMENTS-CLASS in PROCESS-LANGUAGE
  Avenits: (CREATE-STATEMENTS in PROCESS-LANGUAGE ALL NIL)
  Values: FLUID.PHASE.TABLE.INVOCATION

Effect Statement


||| (Output) The FLUID.PHASE.TABLE.EFFECT Unit in THERMAL Knowledge Base

Own slot: STATEMENT from FLUID.PHASE.TABLE.EFFECT
  Inheritance: OVERRIDE.VALUES
  ValueClass: LIST
  PutTranslation: TRANSLATION
  Avenits: (LANGUAGE.TRANSLATE-STATEMENT in PROCESS-LANGUAGE ALL NIL)
  Cardinality.Max: 1
  Comment: This is an actual statement in the Process Language.
  Values: ((OUT1 FLUID.PHASE) <- ((IN1 FLUID.PHASE) PHASE.OP HEAT.GRAD))

Own slot: TRANSLATION from FLUID.PHASE.TABLE.EFFECT
  Inheritance: OVERRIDE.VALUES
  Cardinality.Min: 1
  Cardinality.Max: 1
  Comment: This is the machine translation of the process code.
  Values: (ASSIGNMENT-OPERATOR in PROCESS-LANGUAGE (OUT1 FLUID.PHASE) (PHASE-OPERATOR (IN1 FLUID.PHASE) HEAT.GRAD))

Figure 4 - Thermal Language Operators